Security Implications of Third-Party Accelerators

Lena E. Olson*, Simha Sethumadhavan†, and Mark D. Hill*
*University of Wisconsin-Madison, lena,markhill@cs.wisc.edu
†Columbia University, simha@cs.columbia.edu

Abstract—Third-party accelerators offer system designers high performance and low energy without the market delay of in-house development. However, complex third-party accelerators may include vulnerabilities due to design flaws or malicious intent that are hard to expose during verification. Rather than react to each new vulnerability, it is better to proactively build defenses for classes of attacks. To inspire future work on defenses, this paper develops a taxonomy of accelerator vulnerabilities. We consider the cross product of threat types (confidentiality, integrity, and availability) with risk categories (configuration, computation, termination, accelerator memory accesses, system memory accesses, microarchitecture/coherence, exceptions/interrupts, and power), as well as whether processes can be vulnerable only if they use the offending accelerator (accelerator-scope threat) or even when running in the same system (system-scope threat). Our taxonomy draws attention to a grave problem that needs immediate attention from computer architects.

1 Introduction

Specialized hardware accelerators achieve high performance at low power and energy. Already, many systems contain accelerators for tasks such as video processing or graphics. With an increased market for accelerators, especially in SoC designs, third parties will likely start contributing more to accelerator development.

While the performance and energy benefits of accelerators are fairly well understood, the impact of accelerators, particularly ones designed by third parties, on system security has received less attention. Even so, several security vulnerabilities have already been published for graphics processing units (GPUs), a common class of accelerators. For example, Lee et al. [13] show that GPUs from both NVIDIA and AMD do not clear accelerator memory upon allocation, which allows state from sensitive processes to persist and be read by other, unrelated processes. This allows information leakage between concurrently executing processes or from recently terminated processes. A similar case is discussed by Di Pietro et al. [9]. Some GPU-based malware is designed to take advantage of this uncleared internal memory [20].

These vulnerabilities occurred with non-malicious, in-house development. With outsourced accelerator designs, these problems will likely worsen. It is difficult to ensure that first-party, trusted designs are correct; completely ensuring safety for IP from third-party sources may be harder for a number of reasons, such as time-to-market pressure which makes code reviews infeasible even with source code for the third-party IP.

Security vulnerabilities in accelerators can lead to problems such as data leakage, memory corruption, system crashes, and denial of service attacks. It is particularly worrisome that some security vulnerabilities in accelerators affect processes that are never run on the incorrect accelerator. For example, an accelerator that is allowed to make unrestricted accesses to unified system memory could abuse this by scanning for sensitive data in memory or modifying OS structures. Or it could cause reliability problems by performing wild writes, perhaps caused by stale data stored in the translation lookaside buffer. A bad accelerator could also degrade performance of other system components by unnecessarily saturating shared resources such as memory bandwidth.

Manuscript submitted: 17-Mar-2015. Manuscript accepted: 09-Jun-2015. Final manuscript received: 11-Jun-2015.

How does one defend against accelerator vulnerabilities? A first, reactive approach waits for an exploit to become known, develops a fix in current or future hardware, and repeats for each new exploit. A second, more proactive approach reflects on potential vulnerabilities to seek to mitigate them at design time, and then falls back on the reactive approach. In our judgment, the proactive approach can be enhanced with a systematic taxonomy that can increase coverage of attack classes considered and may inspire defenses that address multiple attack classes.

To this end, we systematically analyze threats posed by the inclusion of accelerators in SoC designs. We aim to make it easier for system designers to make informed, security-conscious choices when integrating accelerators into their designs.

We discuss a number of possible risk categories which may be present in accelerators, as well as their consequences. These consequences can include potential leakage of sensitive information, incorrect computation results, or reduced availability of the accelerator or other system components. We also make the point that accelerator vulnerabilities can be divided into two classes: those that only affect applications running on the vulnerable accelerator, and those that can compromise the entire system. The latter is particularly dangerous, because simply refraining from running sensitive applications on potentially vulnerable hardware is not sufficient to guarantee security.

The main contribution of our paper, a taxonomy of threat space for accelerators, is general enough to cover classes of vulnerabilities rather than forcing designers to consider each possible threat individually, yet also allows threats to be classified based on what components of the system are affected and what security properties they violate. Although we focus on accelerators, many of the threats we discuss are relevant for other types of third-party IP as well.

2 Accelerator vs. CPU Security

There are some practical security differences between accelerators and CPUs. First, CPUs are generally designed by a trusted first party; thus, validation and testing can be done in-house, and the system designer has more assurances that validation has been thorough. In contrast, accelerators may be designed by a third party, and validating them while treating them as a black box is more difficult.

Second, the operating system runs on CPUs, but generally not on accelerators, which has implications for non-malicious accelerators. The presence of the OS allows some security protections to be implemented at the system level rather than in hardware. An example is the way modern OSes clear memory when it is allocated, which protects data from previously executing processes. Without the OS to provide this protection for internal memory in accelerators, we must consider providing it elsewhere, such as in the hardware or at the process level.

There are a large range of accelerators, so there are exceptions to the above. For example, the Xeon Phi is designed by a trusted first party and runs a version of Linux. Thus, some entries in this taxonomy may not apply to every accelerator.

3 Threat Space

We first draw a distinction between accelerators that are malicious by design and those that are buggy, and between threats that affect only the vulnerable accelerator and those that can affect other components of the system.

3.1 Buggy vs. Malicious Accelerators

We classify accelerators with vulnerabilities into two types: *buggy* and *malicious*. Any complex system is likely to have bugs. Bugs occur even in well validated hardware [1], [3], [7], [16], [19]. Buggy accelerators are designed with the intention of being correct, secure accelerators. However, because they are buggy, they have vulnerabilities that can be exploited by unaffiliated parties such as malicious software.

Malicious accelerators are insecure by design: they contain intentionally introduced flaws or are designed to break the security of the host system they are integrated into. As such, malicious accelerators are harder to defend against because the designers can introduce new hardware to aid their exploits and can put effort into hiding incorrect behavior from validation techniques. For example, a malicious accelerator might require a specific uncommon input (a *trigger*) before starting to behave maliciously. One case where accelerators may contain malicious hardware is when the designers are colluding with a government or other body for purpose of surveillance, espionage, or sabotage. There has been concern that hardware trojans might exist in military chips [17].

From the viewpoint of an end user, the distinction between buggy and malicious accelerators is pedantic. After all, end users do not care about why their device is insecure, only that it is insecure. However, as computer designers, addressing these two different threats requires completely different approaches. When addressing a security threat from an honest but buggy device, IP integrators can assume a co-operative third-party IP vendor who will not subvert safety nets meant to prevent bugs. However, for malicious devices, any such safety net should be able to withstand subversion from a malicious IP component.

3.2 Accelerator vs. System Scope

There is an important distinction between security threats that are internal to the affected accelerator and thus only affect applications running on that accelerator, and those that affect other components of the system. An example of a threat at *accelerator scope* is leakage of information between two processes running concurrently on the same accelerator, as can be caused by lack of protection to the accelerator's internal memory. In contrast, threats at *system scope* affect components of the system beyond the compromised accelerator: they also affect correctly implemented parts of the system. An example of a system scope threat is an improper access to a sensitive address in system memory, such as one belonging to the OS.

These two scopes of threats need to be handled differently. It is easier to limit the damage caused by threats at accelerator scope, because they can be avoided by simply not running sensitive applications on untrusted hardware. However, in cases where the designer must treat accelerators as black boxes, it is very difficult to ensure security for applications running on the untrusted accelerator, and sensitive applications may be unable to benefit from acceleration on not fully trusted hardware.

In contrast to threats at accelerator scope, system scope threats are dangerous even to processes that are never run on the untrusted hardware. However, we expect that they can be defended against by careful system design, since they result from a lack of system protection from the accelerators. By placing safeguards such as checking that all interactions between the accelerator and system state are valid, it is possible to limit the damage a malicious accelerator can do to the system.

4 Taxonomy of Threats

We classify security threats for accelerators based on the type of threat they pose as well as what part of the accelerator they affect, as will be summarized in Table 1 (accelerator scope) and Table 2 (system scope).

4.1 Security Threat Types

Security threats can be broken into three types: those that violate confidentiality, integrity, and availability. ***Confidentiality*** violations allow an attacker to discover privileged information; an example is an attacker reading private keys. ***Integrity*** violations cause incorrect results, for example a computation returning the wrong value. ***Availability*** violations prevent a resource from being used; an example is making spurious requests to a resource at such a high rate that legitimate requests cannot be serviced. We consider both threats that make the vulnerable accelerator unavailable and those that reduce availability of other accelerators or the system in general.

4.2 Risk Categories

We break down threats into eight categories, briefly described below. Just as modern pipelined processors fetch, decode, execute, access memory, and write back instructions, accelerators have similar pieplined behavior. The risk categories basically follow this sequence, with the exception of power, which is a cross-cutting concern.

- *Configuration* refers to set up of the accelerator, such as setting registers or giving parameters.
- *Computation* refers to the computations at the accelerator that will produce results.
- *Termination* refers to accelerator state after execution ends.
- *Accelerator Memory Accesses* refers to requests to memory or cache internal to the accelerator.
- *System Memory Accesses* refers to requests to system main memory.
- *Microarchitectural Commands / Coherence Requests* refers to communications about microarchitectural system state between the system and the accelerator, such as memory coherence requests or TLB shootdowns.
- *Exceptions / Interrupts* refers to exceptions that are produced by the accelerator, or interrupts delivered to or generated by the accelerator.
- *Power* refers to threats specifically relating to power consumed by the accelerator.

TABLE 1
Threats at Accelerator Scope. These threats only affect processes running on the untrusted hardware. (x) indicates that a known exploit exists.

	Confidentiality	Integrity	Availability
Configuration	side-channel, kleptography	kleptography, wrong output	lock up accelerator
Computation	side-channel, kleptography	kleptography, wrong output	lock up
Termination	(x) fail to clear registers / memory / cache	stale data in registers / memory / cache	fail to release resources
Accelerator Memory Accesses	(x) bad access	bad access	evict others from memory/caches
System Memory Accesses	side-channel		
Microarchitectural Commands / Coherence Requests		inconsistent (stale) data	
Exceptions / Interrupts	potential timing side-channel		extra exceptions/interrupts
Power	(x) power analysis attacks	excess heat causing unreliability	excessive heat leading to damage

TABLE 2
Threats at System Scope. These threats can also affect processes which are never run on untrusted hardware.

	Confidentiality	Integrity	Availability
Configuration	incorrect registers (e.g. CR3)	incorrect registers (e.g. CR3)	
Termination	stale translations	stale translations	fail to release resources
System Memory Accesses	bad access	bad access	saturate bandwidth or shared caches; cause swapping
Microarchitectural Commands / Coherence Requests	snoop on coherence traffic (side-channel); ignored invalidations	ignore invalidations	excessive coherence requests / ignored coherence requests
Exceptions / Interrupts			extra exceptions/interrupts
Power		high temperature affecting nearby components	high temperatures destroying nearby components

4.3 Detailed Threat Descriptions

We will briefly give examples of vulnerabilities for each of the entries in the threat matrix shown in Tables 1 and 2. We separate out threats at accelerator and system scope, because the risks of each and the approaches to counter them are different. We do not intend to exhaustively list all threats here; different types of accelerators will be vulnerable to different types of threats. We leave some entries blank, but this does not mean that it is impossible for a vulnerability to exist there. However, these tables can be taken as a starting point for considering possible security implications of including an accelerator in the system. For each threat, we indicate whether any exploits are known.

1) Configuration: ***Confidentiality*** Misconfiguration can lead to insufficiently random numbers, wrong modes, or fewer iterations than intended. Kleptography [4], [21], allows texts to be encrypted in such a way that a secret big-brother key can decrypt them. At system scope, a misconfigured CR3 (page table base) register could lead to incorrect page permissions.

Integrity Misconfiguration can result in incorrect results, due to bad initial state.

Availability Some devices may be misconfigured to be undiscoverable by the host.

2) Computation: ***Confidentiality*** Side-channel attacks are well known in CPU processors and caches [5], [6], [11]. They are also possible with accelerators, and certain bugs or design choices can make them easier.

Integrity Accelerators may produce the wrong results of computations. We expect that this will be uncommon, as non-malicious computation errors should usually be caught by validation. However, with a large state space and a complicated accelerator, there is a higher chance of undetected buggy or malicious behavior. furthermore, as shown by the famous Pentium FDIV bug [16], bugs causing incorrect execution can occur even in the presence of validation.

Availability A bug that causes the accelerator to get stuck in a loop or to hang (similar to Halt and Catch Fire (HCF) bugs [3], [7], [19]) will make the accelerator unavailable.

3) Termination: ***Confidentiality*** After program termination, information leakage can occur if internal memories, registers, or other state is not cleared. A known example of this for GPU texture memory exists [9], [13]. In addition, stale address translations could lead to vulnerabilities such as bad system memory accesses.

Integrity Stale address translations or other internal state can lead to incorrect results for computations.

Availability Failure to release accelerator resources prevents other processes from running.

4) Accelerator Memory Accesses: ***Confidentiality*** An accelerator that allows one process running on the accelerator to access accelerator memory owned by another process running on the accelerator can leak information. Di Pietro et al. [9] have demonstrated this exploit on certain GPUs, where kernels of the two processes are interleaved.

Integrity Similarly, an accelerator allowing concurrently executing processes to write to one another's accelerator memory may have correctness errors.

Availability If multiple processes are running concurrently and one is allowed to dominate accelerator resources, the other may suffer from degraded performance. For example, if one process can evict all cache entries belonging to the other, the victim will suffer performance penalties.

5) System Memory Accesses: ***Confidentiality*** An accelerator that can make unrestricted physical memory accesses can read data at addresses unrelated to processes it is running, including operating system memory. This allows reading sensitive data, which can be exfiltrated by writing it to other locations in system memory.

Integrity Similarly, an accelerator can corrupt memory for other running processes, as well as getting incorrect results in computation, by reading and writing to incorrect addresses. Again, this includes the operating system.

Availability An accelerator that permits a high bandwidth of requests to system memory can slow down requests from other system components; for example, an accelerator that erroneously treats some memory accesses as uncacheable.

6) Microarchitectural Commands / Coherence Requests: ***Confidentiality*** An accelerator in a system with a snoopy coherence protocol, where accelerators see coherence requests even for blocks that they do not have access to, could use this information in a side-channel attack, enabling information leakage. Ignoring invalidations or TLB shootdowns can also allow stale translations/blocks to be accessed.

Integrity Stale translations or invalid block accesses, as well as other coherence bugs, can lead to wrong computation results. Malicious coherence requests could cause data corruption in unrelated processes.

Availability Ignored or very slow coherence responses can make the system unresponsive.

7) Power ***Confidentiality*** The power consumed by the accelerator can be used for side-channel attacks, i.e. power analysis attacks [2], [12]. Examples of this type are prevalent, and include attacks against accelerators for AES [14].

Integrity An accelerator may operate at too high power and temperature, for example by switching many transistors at once. High temperature negatively affects reliability and may lead to incorrect computation. This is true not only of the affected accelerator, but also other nearby system components.

Availability An accelerator operating at higher than expected power/temperature can cause overheating in other accelerators or system components, preventing them from running or, in the worst case, causing physical damage.

5 Implications

Some papers provide solutions, while others—like this one—articulate problem spaces to inspire the work of others. To this end, this taxonomy aims to provide a framework to allow accelerator and system designers to carefully consider the security implications of their design choices. We do not—and believe cannot—prove our taxonomy exhaustive, as it does not appear possible to exclude threat classes that one has not thought of (e.g., software researchers might not have considered exploiting coherence to impede availability). Nevertheless, there are several ways this taxonomy can be beneficial.

First, developing a taxonomy can provide a small start toward an accelerator security standard. Currently, the National Institute of Standards and Technology (NIST) publishes the Federal Information Processing Standard (FIPS) for cryptographic modules: FIPS 140-2 [10]. This standard allows certification of cryptographic modules at one of four levels, depending on the protections provided. Although FIPS 140-2 is not exhaustive and does not cover all potential attacks, the certification does provide useful information about the security of cryptographic modules. This taxonomy makes a first step toward developing a similar standard.

Second, the framework we describe allows designers to consider entire classes of potential attacks rather than focusing on vulnerabilities one at a time, and then only once an exploit is discovered. We provide a means of proactively designing defenses, which we hope will lead to fewer vulnerabilities in shipped hardware. Designers will be able to think systematically about threats and defend against them through providing abilities and policies for techniques such as hardware resetting accelerators, monitoring system activity for signs of malicious behavior [8], [15], [18], designing safe interfaces to shared system resources such as memory and cache hierarchies, etc. In addition, the taxonomy and the distinction between internal and system threats allows application and OS developers to systematically evaluate the risks of running sensitive or critical applications on not fully trusted accelerators.

Third, this taxonomy aids in assessing safety and security in a world where, unfortunately, hardware is not guaranteed to be perfect. Instead, recognizing that hardware may be exploitably buggy or even malicious allows us to design defenses and bound the damage that the hardware may do, without requiring that the hardware be flawless.

Acknowledgments

This work is supported through grants FA8750-10-2-0253, FA8650-11-C-7190, NSF 1054844 and the Alfred P. Sloan Foundation. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and may not reflect the views of the funding entities. The authors thank Eric Sedlar, Dan Gibson, Multifacet, and UW-Madison Computer Architecture Affiliates for valuable feedback.

References

[1] *Intel Xeon Processor E5 Family: Specification Update*, Jan. 2014.
[2] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM sidechannel(s)," in *CHES 2002*.
[3] A. D. Balsa, "The Cyrix 6x86 coma bug," http://gwyn.tux.org/%7Ebalsa/linux/cyrix/p11.html, Nov. 1997.
[4] M. Bellare, K. Paterson, and P. Rogaway, "Security of symmetric encryption against mass surveillance," Cryptology ePrint Archive, Report 2014/438, 2014.
[5] R. Callan, A. Zaji, and M. Prvulovic, "A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events," in *MICRO 47*, Dec. 2014.
[6] J. Chen and G. Venkataramani, "CC-Hunter: Uncovering covert timing channels on shared processor hardware," in *MICRO 47*, Dec. 2014.
[7] R. R. Collins, "The Pentium F00F bug," *Doctor Dobb's Journal of Software Tools*, vol. 23, no. 5, pp. 62, 64–66, May 1998.
[8] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters."
[9] R. Di Pietro, F. Lombardi, and A. Villani, "CUDA leaks: Information leakage in GPU architectures," arXiv:1305.7383v2, 2013.
[10] P. FIPS, "140-2: Security requirements for cryptographic modules," *National Institute of Standards and Technology*, 2001.
[11] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *CRYPTO '96*.
[12] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO '99*.
[13] S. Lee, Y. Kim, J. Kim, and J. Kim, "Stealing webpages rendered on your browser by exploiting GPU vulnerabilities," in *SP 2014*.
[14] S. B. Örs, F. Gürkaynak, E. Oswald, and B. Preneel, "Power-analysis attack on an ASIC AES implementation," in *ITCC 2004*.
[15] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *HPCA 21*, Feb. 2015.
[16] D. Price, "Pentium FDIV flaw-lessons learned," *IEEE Micro*, vol. 15, no. 2, pp. 86–88, Apr. 1995.
[17] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *CHES 2012*.
[18] A. Tang, S. Sethumadhavan, and S. Stolfo, "Unsupervised anomaly-based malware detection using hardware features."
[19] G. Wheeler, "Undocumented M6800 instructions," *BYTE Magazine*, vol. 2, no. 12, pp. 46–47, Dec. 1977.
[20] x0r1, "jellyfish," https://github.com/x0r1/jellyfish, 2015.
[21] A. Young and M. Yung, *Malicious Cryptography: Exposing Cryptovirology*. John Wiley & Sons, 2004.